Приложение Г – Полезные программные решения Python

Прежде чем приступать к созданию больших и сложных проектов полезно потренироваться в написании небольших, но эффективных алгоритмов для решения типовых и не совсем типовых задач. Здесь собрано, то, что нам показалось интересным.

#### Простые числа

a, b, p = 10, 1000, [] # кортежу переменных = кортеж присваивания

**if** a % 2 == 0: a += 1

**while** (a < b):

n=0

**for** i **in** range(2, a):

**if** ( i <= a\*\* .5):

**if** (a % i == 0):

n = 1

**else**:

**pass**

**if** (n != 1):

p += [a]

a += 2

print(p, **'\nОбщее количество простых чисел -'**, len(p))

# вариант 2 – простые числа в заданном диапазоне

a = 2; b = 100

p = []

**for** k **in** range(a, b + 1):

prime = **True**

**for** i **in** range(2, k):

**if** k % i == 0:

prime = **False**

**break**

**if** prime:

p += [k]

print(p)

#### Високосные года

Вывести список високосных годов с 1900 по 3000 год.

Years = **''**for Y in range(1899, 3000):  
 if Y % 4 == 0 and Y % 100 != 0 or Y % 400 == 0:  
 Years += str(Y) + **', '**print(Years)

Високосными годами считаются те годы, порядковый номер которых либо кратен 4, но при этом не кратен 100, либо кратен 400 (например, 2000-й год являлся високосным, а 2100-й будет не високосным годом).

#### Проверить число

Проверить число по следующим условиям: 1) равно нулю или нет; 2) положительное или отрицательное; 3) четное или нечетное; 4) дробное или целое.

n = -6.5 # проверяемое число

**if** n == 0:

print(**'Это ноль'**)

**else**:

print([**'Отрицательное'**, **'Положительное'**][n > 0])

print([**'Четное'**, **'Нечетное'**][n % 2 != 0])

print([**'Дробное'**, **'Целое'**][n % 1 == 0])

Здесь вторая часть выражения функции print() возвращает True или False и в зависимости от этого выводится первое или второе значение списка.

#### Формула Герона для нахождения площади треугольника

Здесь переменные a, b и c – длины сторон треугольника.

**def** Heron(a, b, c): *# Формула Герона* p = (a + b + c) \* 0.5  
 **return** (p \* (p - a) \* (p - b) \* (p - c)) \*\* 0.5  
  
  
print(Heron(3, 4, 5))

#### Угадай число

Генератором случайных чисел задано число i. Задача пользователя угадать i последовательно вводя свои варианты чисел g, при этом программа дает подсказки о том, что g больше или меньше чем i.

**import** random *# подключение модуля генерации случайных чисел*i = random.randint(1, 100) *# случайное целое число от 1 до 100*running = **True  
while** running:  
 g = int(input(**'Введите целое число : '**))  
 **if** g == i:  
 print(**'Поздравляю, вы угадали.'**)  
 running = **False** *# это останавливает цикл while* **elif** g < i:  
 print(**'Нет, загаданное число немного больше этого.'**)  
 **else**:  
 print(**'Нет, загаданное число немного меньше этого.'**)  
**else**: *# альтернативный вариант для while*  
 print(**'Цикл while закончен.'**)  
  
print(**'Завершение.'**)

#### Принадлежность точки треугольнику

На плоскости заданы координаты вершин треугольника t. Проверить принадлежит ли точка p данному треугольнику t.

t = [0,0, 3,5, 6,1] *# список координат вершин треугольника*p = [0, 0] *# координаты проверяемой точки*a = (t[0]-p[0]) \* (t[3]-t[1]) - (t[2]-t[0]) \* (t[1]-p[1])  
b = (t[2]-p[0]) \* (t[5]-t[3]) - (t[4]-t[2]) \* (t[3]-p[1])  
c = (t[4]-p[0]) \* (t[1]-t[5]) - (t[0]-t[4]) \* (t[5]-p[1])  
  
**if** (a >= 0 **and** b >=0 **and** c >= 0) **or** (a <= 0 **and** b <= 0 **and** c <= 0):  
 print(**'Точка принадлежит треугольнику'**)  
**else**:  
 print(**'Точка НЕ принадлежит треугольнику'**)

Здесь для решения использовано следующее утверждение – для того, чтобы точка p принадлежала данному треугольнику t, необходимо, чтобы псевдоскалярное (косое) произведение соответствующих векторов было больше или же меньше нуля.

Псевдоскалярное произведение существует только для двумерных векторов, для трехмерного пространства используется тройное скалярное произведение.

**Перевод числа в другую систему счисления**

Основание системы счисления задается переменной base

base = 2  
x = int(input())  
  
**while** x > 0:  
 digit = x % base  
 print(digit, end=**''**)  
 x //= base

#### НОД – алгоритм Евклида

**def** GCD(a, b): #Euclidean algorithm  
 **while** a != 0 **and** b != 0:  
 **if** a > b:  
 a = a % b  
 **else**:  
 b = b % a  
 **return** a + b

#### Вычисления в заданном диапазоне

Вычислить значение функции sin(x), где x вектор значений равномерно распределенных в диапазоне [0; 2π].

**import** numpy **as** np  
print(**'\nВычисление значений в заданном диапазоне'**)  
x = np.linspace( 0, 2 \* np.pi, 12 )  
print(np.sin(x))

#### Решение системы двух линейных уравнений

**import** numpy **as** np  
*# 1\*x1 + 5\*x2 = 11  
 # 2\*x1 + 3\*x2 = 8*a = np.array([[1,5],[2,3]])  
b = np.array([11,8])  
  
print(np.linalg.solve(a,b)) # решение – [1. 2.]

#### Проверка типа числа int с обработкой ошибок

print(**"Type integers, each followed by Enter; or just Enter to finish"**)  
total, count = 0, 0  
**while True**:  
 line = input(**"integer: "**)  
 **if** line:  
 **try**:  
 number = int(line)  
 **except** ValueError **as** err:  
 print(err)  
 **continue** total += number  
 count += 1  
 **else**:  
 **break  
if** count:  
 print(**"count ="**, count, **"total ="**, total, **"mean ="**, total / count)

#### Проверить наличие двух нулей подряд в случайном наборе

**from** random **import** randint

a = [randint(0, 10) **for** i **in** range(int(input(**'Введите количество чисел: '**)))]

b = 0

**for** i **in** range(len(a)):

**if** a[i] == 0 **and** a[i + 1] == 0:

**if** i == len(a):

**break**

b=1

**break**

**if** b == 1:

print(**'Yes'**)

**else**:

print(**'No'**)

#### Перезаписать данные в файл

Открыть файл input.txt, добавить к каждой строке порядковый номер и записать обратно в файл output.txt.

**with** open(**'input.txt'**) **as** fr, open(**'output.txt'**, **'w'**) **as** fw:

**for** i, v **in** enumerate(fr.readlines(), start=1):

print(**f'{**i**} {**v.strip()**}'**, file=fw)

#### Индексы минимального элемента побочной диагонали матрицы

**import** numpy **as** np

a = np.random.randint(-100, 100, size=(6, 6))

b = np.identity(6, dtype=int)

m = 10000;

x = y = 0

**for** n1 **in** range(0, len(a)):

**for** n2 **in** range(0, len(a)):

**if** a[n1, n2] < m **and** b[n1, n2] != 1:

m = a[n1, n2]

x = n1

y = n2

print(a)

print(**'Минимальное число, не стоящее на главной диагонали -'**, m,

**'\nСтрока -'**, x, **'\nСтолбец -'**, y)

#### Числа Фибоначчи

k = 5\*\* .5

**for** i **in** range(21):

*# Формула Бине*

F = int((((1+k)/2) \*\* i - ((1-k)/2) \*\* i) / k)

print(i,**': '**, F)

#### Географические координаты из текста

Функция преобразования текстовой записи географических координат в угловом формате в десятичные числа.

A = **'55°45′07″ с. ш. 37°36′56″ в. д.'**A = **'52°02′ с. ш. 113°30′ в. д.'  
  
  
def** coordinatesDeg(A):  
 **import** re  
  
 res = re.findall(**r'[0-9]+'**, A)  
 res = [int(i) **for** i **in** res]  
 n = int(len(res) / 2)  
  
 lat, lon = 0, 0  
 **for** i **in** range(n):  
 lat += res[i] / (60 \*\* (i % 3))  
 lon += res[i + n] / (60 \*\* (i % 3))  
  
 **return** (lat, lon)  
  
  
print(coordinatesDeg(A)[0])  
print(coordinatesDeg(A)[1])

#### Расстояние между точками на поверхности Земли

Вычислить расстояние по прямой между двумя географическими точками на поверхности Земли по их координатам (долготе и широте) по формуле гаверсинуса (или еще ее называют формула великого круга).

|  |
| --- |
|  |

где:

r – радиус Земли, приблизительно равный 6371 км;

φ и λ – соответственно значения географической широты и долготы.

import numpy as np

# функция coordinatesDeg() определена выше (см. с.5)

lat1, lon1 = coordinatesDeg(A)[0], coordinatesDeg(A)[1]  
lat2, lon2 = coordinatesDeg(B)[0], coordinatesDeg(B)[1]  
  
**def** haversine(lat1, lon1, lat2, lon2):   
 R = 6371 *# радиус Земли* lat1, lon1, lat2, lon2 = map(np.deg2rad, [lat1, lon1, lat2, lon2])  
 dlat = lat2 - lat1   
 dlon = lon2 - lon1   
 a = np.sin(dlat/2)\*\*2 + np.cos(lat1) \* np.cos(lat2) \* np.sin(dlon/2)\*\*2   
 c = 2 \* np.arcsin(np.sqrt(a))   
 total\_km = R \* c   
 **return** total\_km   
  
print(haversine(lat1, lon1, lat2, lon2))

Для проверки были взяты следующие исходные данные и составлена матрица смежности:

| Город А | Координаты города А | Город Б | Расстояние А-Б, км | |
| --- | --- | --- | --- | --- |
| Расчетное | По данным Яндекс |
| Москва | 55°45′07″ с. ш. 37°36′56″ в. д. | Ростов-на-Дону | 957.3 | 959 |
| Ростов-на-Дону | 47°14′26″ с. ш. 39°42′38″ в. д. | Владивосток | 6796.8 | 6798 |
| Чита | 52°02′ с. ш. 113°30′ в. д. | Москва | 4736.7 | 4737 |
| Владивосток | 43°07′ с. ш. 131°54′ в. д. | Москва | 6416.4 | 6417 |

Как видно разница между расчетными значениями и данными Яндекс не превышает двух километров, что очень даже не плохо. Для населенных пунктов в Яндекс и большинстве других сервисов указываются координаты местонахождения главного административного здания.

#### Поиск наибольших / наименьших значений

**import** heapq # Пример из книги Д.Бизли «Python. Книга рецептов» [1]  
  
nums = [1, 8, 2, 23, 7, -4, 18, 23, 42, 37, 2]  
print(heapq.nlargest(3, nums)) *# Выведет [42, 37, 23*print(heapq.nsmallest(3, nums)) *# Выведет [-4, 1, 2]*

Обе функции также принимают параметр key, который позволяет использовать их с более сложными структурами данных.

**import** heapq  
  
portfolio = [  
 {**'name'**: **'IBM'**, **'shares'**: 100, **'price'**: 91.1},  
 {**'name'**: **'AAPL'**, **'shares'**: 50, **'price'**: 543.22},  
 {**'name'**: **'FB'**, **'shares'**: 200, **'price'**: 21.09},  
 {**'name'**: **'HPQ'**, **'shares'**: 35, **'price'**: 31.75},  
 {**'name'**: **'YHOO'**, **'shares'**: 45, **'price'**: 16.35},  
 {**'name'**: **'ACME'**, **'shares'**: 75, **'price'**: 115.65}  
]

cheap = heapq.nsmallest(2, portfolio, key=**lambda** s: s[**'price'**])

expensive = heapq.nlargest(2, portfolio, key=**lambda** s: s[**'price'**])

print(cheap)

print(expensive)

В первом случае будут выведены два самых дешевых, а во втором два самых дорогих словаря.

### Комбинаторика

Комбинация – это набор элементов, порядок которых не имеет значения.

Перестановка – это расположение набора, в котором порядок имеет значение.

Функция permutations() из модуля itertools вычисляет количество перестановок элементов множества.

**import** itertools **as** it  
  
print(**'Количество перестановок из n по m без повторений'**)  
n, m = **'12345'**, 3  
**for** count, val **in** enumerate(it.permutations(n, m), start=1):  
 print(count, **'- '**, end=**''**)  
 **for** i **in** val:  
 print(i, end=**''**)  
 print()

# следующий код выводит аналогичные результаты в виде кортежей

print(\*it.permutations(n, m))

Алгебраическая форма записи, приведенного выше кода:

print(**' Прямое, или декартово произведение двух множеств'**)  
n, m = **'12345'**, **'abc'**  
**for** count, val **in** enumerate(it.product(n, m), start=1):  
 print(count, **'- '**, end=**''**)  
 **for** i **in** val:  
 print(i, end=**''**)  
 print()

Результат декартового произведения упорядоченные пары элементов из первого и второго множеств.

print(**'Количество сочетаний из n по m с повторениями'**)  
n, m = **'12345'**, 3  
**for** count, val **in** enumerate(it.product(n, repeat=m), start=1):  
 print(count, **'- '**, end=**''**)  
 **for** i **in** val:  
 print(i, end=**''**)  
 print()

Сочетания без возвратов вычисляются функциями combinations, алгебраическая запись формулы:

print(**'Количество сочетаний из n по m без возвратов '**)  
n, m = **'12345'**, 3  
**for** count, val **in** enumerate(it.combinations(n, m), start=1):  
 print(count, **'- '**, end=**''**)  
 **for** i **in** val:  
 print(i, end=**''**)  
 print()

Сочетания с возвратами элементов вычисляются функцией combinations\_with\_replacement, алгебраическая запись формулы:

print(**'Количество сочетаний из n по m с возвратами '**)  
n, m = **'12345'**, 3  
**for** count, val **in** enumerate(it.combinations\_with\_replacement(n, m), start=1):  
 print(count, **'- '**, end=**''**)  
 **for** i **in** val:  
 print(i, end=**''**)  
 print()

### Обработка текста

#### Количество гласных букв в строках

a = 'аиеёоуыэюя'

s = ('Багровый и белый отброшен и скомкан, ',

'в зеленый бросали горстями дукаты, ',

'а черным ладоням сбежавшихся окон ',

'раздали горящие желтые карты.',

'В. Маяковский')

**for** i **in** s:  
 print(i,**'-'**, sum([1 **for** n **in** i **if** n **in** a]),**'гласных'**)

Оказывается в каждой строке по 12 гласных.

#### Удалить из текста лишние символы по списку

s = ('Мороз и солнце; день чудесный!')

p = '.,;-—!:…+?' # список лишних символов

for i in p:

s = s.replace(i, '')

#### Палиндром

Палиндром – число, буквосочетание, слово или текст, одинаково читающееся в обоих направлениях. Например, число 101; слова «топот» в русском языке и «saippuakivikauppias» в финском (продавец мыла) – самое длинное слово-палиндром в мире; строки «А, роза упала на лапу Азора!», «Несун гнусен» и пр.

Представленная далее программа проверяет, является ли введенный текст палиндромом. Важным условием является предварительное удаление из текста всех пробелов и знаков препинания.

**def** is\_palindrome(tx):  
 t1 = **''**.join(filter(str.isalnum, tx)).lower()  
 **return** t1 == t1[::-1]  
  
s = **'А, роза упала на лапу Азора!'**print(is\_palindrome(s))

#### Наиболее часто встречающиеся элементы

Используется метод Counter из модуля collections

from collections import Counter

car = [

'АЦ', 'ПНС', 'АЦ', 'АЛ', 'АЛ', 'АШ', 'ПНС', 'АШ', 'АШ', 'АЦ', 'АЦ', 'АЦ',

'АР', 'АЦ', 'АР', 'АЦ', 'АР', 'ПНС', 'АР', 'АЦ', 'АЦ', 'АЛ', 'АШ', 'АР',

'АЛ', 'АР', 'АР', 'АР', 'ПНС', 'АЛ', 'АЦ', 'ПНС', 'ПНС', 'АЦ', 'АЛ', АЛ',

'АЛ', 'ПНС', 'АЦ', 'АШ', 'АШ', 'АЦ'

]

car\_counts = Counter(car)  
top\_car = car\_counts.most\_common(6)  
print(top\_car) # результат: [('АЦ', 13), ('АЛ', 8), ('АР', 8)]

# количество указанных элементов

print(car\_counts[**'АШ'**]) # результат: [('АШ', 6)

# увеличение количества указанных элементов

car\_counts[**'АШ'**] += 10  
print(car\_counts[**'АШ'**]) # результат: [('АШ', 16)

# пересчитать первых три элемента по частоте встречаемости  
top\_car = car\_counts.most\_common(3)  
print(top\_car) # результат: [('АШ', 16), ('АЦ', 13), ('АЛ', 8)]

# добавление дополнительного списка элементов  
car1 = [  
**'АЦ'**,**'ПНС'**,**'АЦ'**,**'АЛ'**,**'АЛ'**,**'АШ'**,**'ПНС'**,**'АШ'**,**'АШ'**,**'АЦ'**,**'АЦ'**,**'АЦ'**,**'АР'**,**'АЦ'**,**'АР'**,**'АЦ'**]

car1\_counts = Counter(car1)  
top\_car1 = car1\_counts.most\_common(3)  
print(top\_car1, ' - car1') # результат: [('АЦ', 7), ('АШ', 3), ('ПНС', 2)] - car1

# добавление дополнительного списка элементов циклом

for i in car1:  
 car\_counts[i] += 1  
print(car\_counts)

# добавление дополнительного списка элементов с помощью метода update

car\_counts.update(car1)  
print(car\_counts, ' - .update(car1)')

*# объединение счетчиков*a = Counter(car)  
b = Counter(car1)  
c = a + b  
print(c)  
  
*# вычитание счетчиков*d = a - b  
print(d)

#### Количество знаков пунктуации

target = [**';'**,**'.'**,**','**,**'–'**,**'!'**]  
string = **"Природа Камчатки и удивительные ее вулканы - это чрезвычайно изменчивое и красочное зрелище. Человека, которому удалось стать свидетелем извержений, навсегда завораживает мощная стихия огня, потоки раскаленной лавы, бомбовые залпы, фейерверки огненных камней! Естественно, фотографии и фильмы, посвященные вулканам и их извержениям, имеют огромную познавательную ценность. В последние годы была проведена новая серия съемок камчатских вулканических ландшафтов. Они дополняли аэросъемки, проводившиеся в течение пяти лет. Повторение систематической съемки вулканов Камчатки позволило увидеть, как быстро изменяется вулканический рельеф."**

n = 0  
**for** punct **in** target:  
 **if** punct **in** string:  
 n += string.count(punct)  
print(n) # 16

Обратите внимание на метод count(), который возвращает суммарное количество определенного знака в тексте. Если вместо count() написать выражение суммирования n += 1, то результат будет 4 – это количество видов знаков пунктуации в данном тексте.

#### Разобрать адрес сайта с помощью регулярных выражений

**import** regex **as** re  
adr = [**'http://somesite.com:80/index.html'**,  
 **'ftp://filerepo.com:21/arch.zip'**]  
  
**for** i **in** adr:  
 print(i)  
 print(**'Protocol: '**, re.search(**r'\b\w+'**, i)[0])  
 print(**'Domain: '**, re.search(**r':\/\/(.+):'**, i).group(1))  
 print(**'Port: '**, re.search(**r':(\d+)'**, i).group(1))  
 print(**'File name: '**, re.search(**r'(\w+)\.\w+$'**, i).group(1))  
 print(**'File type: '**, re.search(**r'\w+$'**, i)[0])  
 print()

Аргумент метода group() указывает какую именно скобочную группу нужно выбирать. Если аргумент опущен, то возвращается полностью все искомое выражение. Знак доллара указывает на то, что поиск ведется, начиная с конца строки.

Далее выполняется тоже самое, только шаблон поиска задан одной строкой с выделением групп, которые затем методом format вставляются в указанные места строки.

**import** regex **as** re  
  
adr = [**'http://somesite.com:80/index.html'**, **'ftp://filerepo.com:21/arch.zip'**]  
  
**for** addr **in** adr:  
 res = re.search(**r'(\w+)://(\w+\.\w+):(\d+)/(\w+)\.(\w+)'**, addr)  
 print(**'Protocol: {}\nDomain: {}\nPort: {}\nFile name: {}\nFile type: {}'**.format(\*res.groups()))

#### Очистить текст от непечатных символов

ts = **'The quick brown fox\njumps\tover the\tlazy dog\r\n'**ch\_map = {  
 ord(**'\n'**): **' '**,  
 ord(**'\t'**): **' '**,  
 ord(**'\r'**): **None**}  
  
print(ts)  
print(ts.translate(ch\_map))

#### Рекурсия

Пример вывода числа в обратном порядке с использованием рекурсии функции.

**def** foo(n):

if n > 0:

print(n % 10)

foo(n // 10)

foo(int(input()))

### Список (list)

#### Двоичный поиск и поддержание списка в отсортированном состоянии

Функции bisect() и insort() стандартного модуля bisect позволяют поддерживать предварительно отсортированный список в упорядоченном состоянии. Параметры функций:

– список или имя списка;

– вставляемый элемент;

– опционально можно указать диапазон поиска, например lo=0, hi=5.

**import** bisect  
  
c = [1, 2, 2, 2, 3, 4, 7]  
bisect.bisect(c, 2) # 4 – позиция в списке для вставки новой двойки после всех уже имеющихся в списке двоек  
bisect.bisect(c, 5) # 6 – позиция в списке для вставки новой пятерки между четверкой и семеркой  
bisect.insort(c, 6) # вставка шестерки после четверки

Функции модуля bisect – не выполняют проверку сортировки списка.

#### zip() – объединение элементов нескольких списков

Функция zip() последовательно «сшивает» элементы нескольких списков.

a, b, c = list(**'first'**), list(**'second'**), list(**'thirds'**)  
z = zip(a, b, c)  
print(list(z))

# результат:

[('f', 's', 't'), ('i', 'e', 'h'), ('r', 'c', 'i'), ('s', 'o', 'r'), ('t', 'n', 'd')]

Минимальная длинна списков a, b, c – шесть элементов и поэтому функция zip() вернула шесть троек из букв. Кроме того функция zip() позволяет вернуть «сшитый» список в исходное состояние.

pitchers = [(**'Nolan'**, **'Ryan'**), (**'Roger'**, **'Clemens'**), (**'Schilling'**, **'Curt'**)]  
first\_names, last\_names = zip(\*pitchers)  
print(first\_names)  
print(last\_names)